(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
30 August 2001 (30.08.2001)

PCT

(10) International Publication Number
WO 01/63852 A1

(51) International Patent Classification[7]: H04L 12/56

(21) International Application Number: PCT/FI01/00163

(22) International Filing Date: 20 February 2001 (20.02.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
20000396 21 February 2000 (21.02.2000) FI

(71) Applicant *(for all designated States except US)*: **TELLABS OY** [FI/FI]; Sinikalliontie 7, FIN-02630 Espoo (FI).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **HEIKKILÄ, Jukka** [FI/FI]; Tullimiehentie 1 B 7, FIN-90560 Oulu (FI). **TAKALA, Mikko** [FI/FI]; Tuulikintie 3 F 110, FIN-90570 Oulu (FI).

(74) Agent: **BERGGREN OY AB**; P.O. Box 16, FIN-00101 Helsinki (FI).

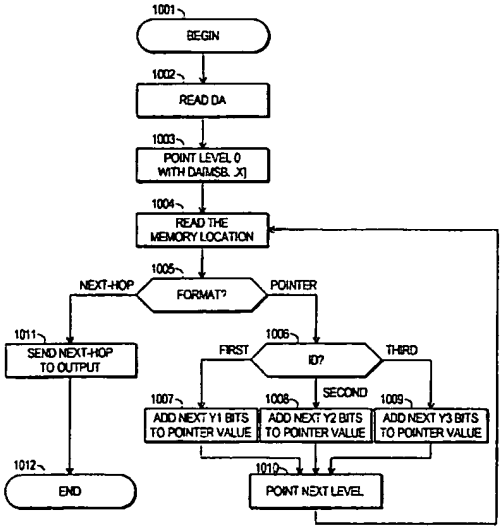
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*
— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

[Continued on next page]

(54) Title: A METHOD AND ARRANGEMENT FOR CONSTRUCTING, MAINTAINING AND USING LOOKUP TABLES FOR PACKET ROUTING



(57) Abstract: A method is disclosed for searching through a variable stride level compressed search tree the nodes of which are represented by records in a memory. The stride length (402, 404) to be used in advancing to a certain lower level from a certain node in the search tree is read from the record (502, 601, 603) that represented said node. A method is also disclosed for constructing a variable stride level compressed search tree the nodes of which are represented by records in a memory. The construction method comprises the step of storing the stride length (402, 404) to be used in advancing to a certain lower level from a certain node in the search tree into the record that represents said node. Additionally there are disclosed a device and a router for implementing the searching and construction methods.

WO 01/63852 A1

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

A method and arrangement for constructing, maintaining and using lookup tables for packet routing

The invention concerns generally the technology of routing packets through a packet-switched data network. Especially the invention concerns the technology of constructing, maintaining and using lookup tables for the routing task so that the number of memory accesses and the amount of required memory are as small as possible.

Figs. 1 and 2 illustrate schematically some aspects of the known task of routing a data packet 101 from a sender node 102 to a recipient node 103 (or sender sub-network to recipient sub-network) through a packet-switched data network 100 consisting of a number of interconnected routers. For the sake of simplicity only routers 104, 105, 106 and 107 are shown in Fig. 1; practical packet-switched data networks may comprise thousands of routers. Fig. 2 shows schematically a router 104 with a number of input ports 201, 202 and 203, a number of output ports 211, 212 and 213, a cross-connecting block 220 capable of coupling a packet from an arbitrary input port to any of the output ports, a control block 221 arranged to control the cross-connecting functionality of block 220 and a memory 222 coupled to the control block. In practice a router does not comprise physically separate connections for the input ports and output ports; Fig. 2 only serves to illustrate the concept of connecting between ports.

The data packet 101 comprises, in a destination address or DA field (not separately shown in Fig. 1), a destination address of the recipient node 103. The destination address consists of a network prefix and a host identifier. The router 104 maintains, in the memory block 222, a look-up table (not separately shown in Fig. 2) where recipient addresses are associated with output port identifiers. The network prefix of the recipient node 103 appears in the lookup table together with the identifier of that one of the output ports 211, 212 or 213 into which all packets to that recipient should be coupled in order to correctly pass them along on their way from the sender to the recipient. When a packet comes into the router 104 through one of the input ports 201, 202 or 203, the control block 221 reads the destination address from the packet and uses it as a search key to find the corresponding output port identifier from the memory 222. It gives the output port identifier it found to the cross-connecting block 220 as a cross-connecting instruction so that the latter is able

to take the packet and couple it into the correct output port. The output port identifier is also known as the next-hop.

The routing arrangement should be optimized regarding both speed and memory consumption. A specific feature of the IP or Internet Protocol, which at the priority date of this patent application constitutes the most important framework for routing applications, is that the length of the DA field is fixed but the length of the network prefix is not, and the search algorithm must find the longest possible match in the look-up table. Several known approaches exist for devising the necessary algorithms for constructing, maintaining and using the look-up table. In this patent application we concentrate on level compressed (LC) search tree approaches.

Figs. 3a, 3b and 3c illustrate the search for five given variable length addresses in a binary tree (Fig. 3a), an LC tree with 3 levels (Fig. 3b) and an LC tree with 2 levels (Fig. 3c). In each tree the exemplary addresses 00, 0101, 111000, 111010 and 1111 should be found. Searching begins at the root of the tree and follows the nodes downwards until a (longest) match is found. In the binary tree of Fig. 3a the search for address 00 terminates at node 301, the search for address 0101 at node 302, the search for address 111000 at node 303, the search for address 111010 at node 304 and the search for node 1111 at node 305. In Fig. 3b the corresponding terminating nodes are 311, 312, 313, 314 and 315. In the two-level LC tree of Fig. 3c several terminating nodes give a match for those addresses where the number of bits is not a multiple of 3. Nodes 321a and 321b match address 00, nodes 322a, 322b, 322c and 322d match address 0101, node 323 matches address 111000, node 324 matches address 111010 and nodes 325a, 325b, 325c and 325d match address 1111.

Binary trees are ineffective for searching because the number of levels in the tree correlates directly with the number of memory accesses required to perform a search; a large number of memory accesses means slowness in performance. Level compressing decreases the number of required memory accesses but increases the number of matching nodes in the search tree, which correlates more or less directly with the required amount of memory. Note that dashed-line nodes in Figs. 3a, 3b and 3c do not refer to validly stored routing information.

The number of bits that are considered in looking for the next matching node in a search tree is known as the stride. The trees in Figs. 3a, 3b and 3c all apply a constant stride, which is 1 for Fig. 3a, 2 for Fig. 3b and 3 for Fig. 3c. Fig. 3d shows how the known principle of variable stride can be applied to limit the number of matching nodes. In the subtree on the left under node 330 we either have to use

stride 3 as in Fig. 3d or accept the cost of increasing the number of levels in the tree. However, in the subtree on the right under node 331 we may apply stride 1 instead of 3, which cuts down the number of nodes matching address 0101 from four to one (node 322).

Using variable stride means that the task of constructing and maintaining search trees become more complex. The length of the stride needs to be stored in memory separately for each branch. Reading the stride length from its storage location requires a separate memory access, which tends to slow down the search and make the known variable stride techniques unattractive for fast routing purposes.

It is an object of the invention to provide a method and an arrangement for constructing, maintaining and using look-up tables quickly and effectively for routing purposes. It is a further object of the invention that the method and arrangement are relatively easy to implement and easily scalable to different sizes of look-up tables. It is an additional object of the invention that the method and arrangement are not heavily device dependent.

The objects of the invention are achieved by using variable stride length in a look-up table and placing the stride length of a branch into a same memory entry with other information concerning the branch.

The method according to the invention for searching through a variable stride level compressed search tree the nodes of which are represented by records in a memory is characterized in that the stride length to be used in advancing to a certain lower level from a certain node in the search tree is read from the record that represented said node.

The method according to the invention for constructing a variable stride level compressed search tree the nodes of which are represented by records in a memory is characterized in that it comprises the step of storing the stride length to be used in advancing to a certain lower level from a certain node in the search tree into the record that represents said node.

The invention applies also to a device for searching through a variable stride level compressed search tree, comprising a memory and, stored within the memory, records that represent the nodes of the search tree. The device according to the invention is characterized in that the memory comprises within at least a number of the stored records an integral part of the record that indicates a stride length to be

used in advancing to a certain lower level from a certain node in the search tree represented by the record.

Additionally the invention applies to a router for constructing and using a search tree for retrieving pieces of routing information concerning the further treatment of data packets on the basis of pieces of address information contained within the packets, comprising
- a microprocessor
- a bus coupled to said microprocessor,
- a search tree module coupled to said bus,
- within the search tree module a memory circuit;
the router according to the invention is characterized in that the microprocessor is arranged to
a) group the pieces of routing information into subtree lists so that within each subtree list the pieces of routing information share a number of equally valued most significant bits,
b) represent the equally valued most significant bits of the pieces of routing information within a subtree list with an inner node of the search tree to be constructed,
c) select a stride length to be applied in advancing to the next lower level from the inner node and
d) place the pieces of routing information within the subtree list into a subtree under the inner node, the number of branches in the subtree being determined by the selected stride length;
and that the microprocessor is further arranged to store the selected stride length in the form of an indicator value into the record that represents the inner node within the memory circuit.

According to the invention the look-up table where next-hops are to be found consists of a relatively limited number of subtables or logically separate entities which correspond to the levels of a variable stride level compressed (VSLC) search tree. A fixed stride, i.e. a fixed number of bits from a piece of address information in a packet, is used on the highest level as a pointer to a certain memory location within a first subtable. This memory location contains either a next-hop or a pointer to a second subtable, which is a memory block on the next lower level. Additionally the memory location contains an identifier that indicates the contents of the memory location to be either a next-hop or a pointer to the next lower level. If the contents of the memory location are identified as a pointer, the value of the identifier also

tells the stride length to be applied on the next lower level. The stride length is the number of additional bits that have to be taken from the piece of address information in order to find a memory location within the memory block pointed at by the pointer. This new memory location may again contain either a next-hop or a pointer to a memory block (with the associated stride length information) on the next, even lower level. The chain of pointers is followed, with more additional bits taken every time from the piece of address information, until a next-hop is found.

A given list of mappings between network addresses and next-hops may be implemented in a number of mutually alternative search trees, when variable stride lengths are used. Constructing and maintaining an optimal look-up table involves experimenting with different stride lengths, giving rise to different subtree structures, until a solution with optimal properties is found. The optimization algorithm may be weighted to always give the minimum achievable number of levels, to always give the minimum achievable number of leaves in the search tree, or to compromise between these two objectives according to some predetermined rules.

According to an advantageous embodiment of the invention the look-up table construction algorithm is recursive so that it constructs and optimizes all subtrees under a certain parent node, calling itself again at each child of the parent node, before moving on in the search tree structure. For each subtree the algorithm tries all allowable stride lengths and selects the one which enables the construction of a most optimal subtree. For example, if the search tree structure is to be optimized for the minimum memory consumption, the construction algorithm returns the subtree (and the associated stride length) with the minimum number of leaves.

An advantageous hardware implementation of the invention relies on the use of static random access memories or SRAMs or other applicable memory devices where the actual next-hops and pointers are stored. A piece of address information is first read into an input register from which a fixed number of most significant bits are directly taken as a memory address to a first memory device. The contents from the memory location pointed at by said memory address are taken into a controller which investigates the identifier contained therein to decide, whether the rest of the contents should be interpreted as a pointer or a next-hop. The controller either directs the rest of the contents into an output register or takes a number of additional bits from the input register, adds them to the rest of the contents from the first memory location and uses the result as a memory address to a second memory device, which physically may be the same as the first memory device or another

memory device. Pipelining the operations, using different memory devices, typically results in improved performance.

The novel features which are considered as characteristic of the invention are set forth in particular in the appended claims. The invention itself, however, both as to its construction and its method of operation, together with additional objects and advantages thereof, will be best understood from the following description of specific embodiments when read in connection with the accompanying drawings.

Fig. 1 illustrates a known network arrangement,

Fig. 2 illustrates schematically the known operation of a router,

Figs. 3a to 3d illustrate known search tree structures,

Figs. 4a to 4d illustrate advantageous uses of a record according to an embodiment of the invention,

Fig. 5 illustrates the application of an embodiment of the invention to an exemplary search,

Fig. 6 illustrates the application of an embodiment of the invention to another exemplary search,

Fig. 7 illustrates a method according to an embodiment of the invention,

Fig. 8 illustrates an arrangement according to an embodiment of the invention,

Fig. 9 illustrates a detail of the arrangement in Fig. 8 and

Fig. 10 illustrates a method according to another embodiment of the invention.

Search trees are most advantageously implemented so that each node (where "node" means both the inner nodes and the leaves of the tree) is represented by a record in a memory. A record is the smallest logical unit in a memory that can be unequivocally pointed at through a memory address. If the memory is physically thought to consist of separate memory locations, a record is exactly large enough to fill a memory location.

In routing applications the piece of information ultimately looked for is a next-hop. A record in the search tree structure of a look-up table comprises either a next-hop, in which case the record represents a leaf of the search tree, or a pointer to some further memory location. In the latter case the record represents an inner node of the search tree.

According to the invention a record which comprises a pointer may also comprise information regarding the stride length that should be applied when the search advances by following the pointer to the further memory location. Fig. 4a illustrates a record 400 which consists of three fields: a part field 401, a stride field 402 and a pointer or next-hop field 403. The value in the part field 401 indicates whether the record represents a part of a route through the search tree, i.e. an inner node, or whether it represents a leaf. If the record represents a part of a route, the value in the stride field 402 indicates the stride length that should be applied when the search advances. The third field 403 comprises either a pointer or a next-hop.

Fig. 4b illustrates a generalization of the record of Fig. 4a. Here the record 400 is of the same size as in Fig. 4a, but the part and stride fields have been replaced with a common record identifier field 404. A value in this field may be used for both of those purposes which were given above to the part and stride fields. The generalization is advantageous because it may help to keep the length of the field(s) other than the pointer or next-hop field 403 short, thus saving bits to "payload" use. As an example we may consider a situation where there are three possible stride lengths. If the part and stride fields are independent of each other (as in Fig. 4a), at least one bit is needed for the part field and at least two for the stride field. However, taken the record of Fig. 4b we may make the following definitions for the common record identifier field 404:

Table 1

field value	meaning
00	record contains a next-hop
01	record contains a pointer, first predefined stride length
10	record contains a pointer, second predefined stride length
11	record contains a pointer, third predefined stride length

This way only two bits are required for the same indications as above.

Some records may not need to comprise the stride length information at all. Fig. 4c illustrates a record 400 that comprises only the part field and a pointer or next-hop field 405 which is now slightly longer than that of Figs. 4a and 4b. Examples of records where the stride length information is not needed are those that represent leaves (the value in the part field 401 tells that the record represents a leaf) or those that belong to such a level in the search tree where the next applicable stride length is constant for some reason or another.

If the maximum number of levels in the search tree is fixed and the search algorithm is able to track the number of levels it has advanced through, the records that represent nodes on the lowest level need not comprise part or stride information at all: all nodes on the lowest level are leaves and the search won't advance further. Fig. 4d illustrates a record 400 which consists of a single long next-hop field 406.

Next we will describe an exemplary search according to an embodiment of the invention with reference to Fig. 5. A destination address 501 has been read from a packet. In Fig. 5 the bits of the destination address appear in a column with the most significant bit (MSB) at the top. As an example we may think that the destination address 501 is an IPv4 (Internet Protocol version 4) address so that it consists of 32 bits, numbered from 31 to 0 with the MSB first. In the destination address 501 shown in Fig. 5, bits 31-8 constitute a network identifier and bits 7-0 constitute a host identifier.

Fixed stride length N, here 16, is used at level 0 which is the highest level in the search tree. The N most significant bits, here bits DA[31-16], are extracted from the destination address 501 and used as a memory address to identify a first memory location 502. A record of the type shown in Fig. 4b is read from the first memory location 502. A value "11" appears in the common record identifier field. Here we assume that the definitions given in Table 1 apply, with 8 bits as the third predefined stride length. The pointer P0 is therefore read from the pointer or next-hop field of the record, and eight additional bits (here bits DA[15-8]) are taken from the destination address 501.

The pointer P0 and the additional bits are used together at level 1 which is the next lower level in the search tree. The pointer P0 as such identifies a memory block 503 which contains as many memory locations that can be separately identified with as many bits as there are in the stride. Here the stride length is 8, so memory block 503 contains 256 memory locations. Of these, the eight additional bits extracted from the destination address 501 point at a certain single memory location 504. A simple

way to implement the combined use of the pointer P0 and the additional bits DA[15-8] is to make the pointer P0 give the address of the memory location at the beginning of the block, so that the address of the memory location 504 is obtained simply by summing the value of the pointer P0 and the additional bits DA[15-8].

In Fig. 5 the value "00" appears in the common record identifier field of the record read from memory location 504. According to Table 1, this means that the record contains a next-hop and the search has terminated. The search algorithm returns the next-hop N-H 1 as the result of the search.

Fig. 6 illustrates an exemplary search through a search tree with three levels. Now we assume that the definitions of Table 1 apply so that the first predetermined stride length is three and the second predetermined stride length is five. At level 0 a fixed stride length of 16 bits is applied, and exactly as in Fig. 5 a first memory location 601 is identified and a record is read therefrom. This time the common record identifier field has a value "01", which means that the record contains a pointer and that three additional bits should ne extracted from the destination address 501. At level 1 the pointer P0 points at the beginning of a memory block 602 which now contains only eight memory locations, since the current stride (3 bits) only allows for eight memory locations to be unambiguously identified. Adding the bits DA[15-13] to the value of the pointer P0 identifies a second memory location 603, from which another record is read.

This time the value of the common record identifier field is "10", meaning that also this record contains a pointer (designated as pointer P1) and that five additional bits should ne extracted from the destination address 501; stride length is five. At level 2 the pointer P1 points at the beginning of a memory block 604 which contains only 32 memory locations. Adding the bits DA[12-8] to the value of the pointer P1 identifies a third memory location 605.

In Fig. 6 we have assumed that level 2 is the lowest possible level in the search tree. Therefore the record read from the third memory location 605 is of the type shown in Fig. 4d: it only contains a next-hop without any part, stride or common record identifier fields. The search algorithm returns the next-hop N-H 2 as the result of the search.

Next we will address the task of constructing the search tree so that it can be used in the manner described above. We designate the destination addresses or parts thereof that should be placed into the search tree as routes for short. In Fig. 7, after

beginning at step 701, the fixed stride length of the highest level is taken at step 702. The construction algorithm groups the routes into subtree lists so that all those routes come onto the same subtree list which are equal by as many most significant bits as the stride length indicates. Previously we assumed that the fixed stride length of the highest level is 16, so the loop consisting of steps 703, 704, 705 and 706 returns a maximum of 65536 subtree lists where each subtree list is characterized by the 16 most significant bits of all routes therein being the same.

The idea of the exemplary construction algorithm shown in Fig. 7 is to take one subtree at a time and to find an optimal form for it in terms of minimum memory consumption. Within each subtree the branches thereof are further treated as sub-subtrees which are optimized one by one. At step 707 the algorithm takes a certain k:th subtree, and at step 708 it takes a certain p:th stride. In order to examine, whether the subtree under construction will need still lower levels, the algorithm subtracts the stride length from all routes of the subtree at step 709. If longer routes are found at step 710, the construction algorithm calls itself which means going down one level at step 711, composing new route lists from those routes that were too long to fit on the previous level at step 712 and returning to step 707. If no longer routes are found at step 710, the construction algorithm constructs the subtree with the stride selected at step 708 and returns its size. In order to find the subtree structure with minimum size, all allowable strides are experimented with by jumping back from step 714 through the stride index incrementing step 715 to step 708 until all allowable strides have been tried.

A typical search tree construction algorithm ensures that no node in the tree will become simultaneously an inner node and a leaf. In other words, it is typically not admissible that e.g. with a stride length 3, there would be a directly matching route 111 (a leaf) and under it a further subtree for matching routes like 111001 or 111010. Several approaches may be used to avoid such situations.

A first alternative approach is to avoid cases of the above-mentioned kind by choosing either a shorter or longer stride: for example the algorithm might choose first a stride length 2 (if available), resulting an inner node for a bit pattern 11, and after that some other stride length which would make a difference between the routes. Or the algorithm might choose a longer stride like 6, reducing the need for additional levels but increasing the number of leaves.

A second alternative approach is to use the empty leaves in the subtree under a matching node to store the next-hop for the route that actually matches the root node

of the subtree. If we take our examples of matching routes 111, 111001 and 111010, a stride length 3 would bring us to the root node 111. This node would not in this approach contain the next-hop but a pointer to the next lower level. Taken that stride length 3 is applied also on that level, of the eight leaves in the subtree the leaves 001 and 010 contain the next-hops for the matching routes 111001 and 111010 respectively and all other leaves contain the next-hop for the matching route 111.

The smallest subtree is selected and the others are discarded at step 716. At step 717 the construction algorithm checks whether such nodes still exist that are siblings to the root of the finished subtree and could become subtree roots themselves. Incrementing the subtree index k at step 718 means selecting the next subtree on the same level for construction and optimization. At step 717 the algorithm eventually finds that all subtrees of a certain level have been constructed, so a check is made at step 719 whether there are still untreated branches starting from the next higher level. A positive finding leads into going up one level at step 720 and incrementing the subtree index k of that level at step 718 before starting the actual subtree construction from step 707. A negative finding at step 719 means that the whole search tree is complete and the construction algorithm terminates at step 721.

If other optimization criteria are to be taken into account than just the size of a subtree, these are easily added into the check illustrated as step 716 in Fig. 7. The search tree construction algorithm always selects at step 716 that subtree that matches best a predetermined set of optimization criteria.

Maintaining a search tree means that when a route is removed or a new route is added, in addition to the actual storing or deleting of a piece of routing information a check must be made whether the optimality of a previously constructed search tree still holds. Re-optimization does not need to cover the whole search tree. It suffices to re-optimize from the leaf which was affected by the change upwards to that subtree level where the re-optimization does not change a previously selected stride length. In all cases it is certain that not more that that subtree needs to be re-optimized the root of which is at the highest level and the domain of which was affected by the change.

In order to find the optimal search tree with all its optimized subtrees the construction algorithm must make a considerable number of trial and error rounds with different stride lengths. Increasing the number of allowable stride lengths and the maximum number of levels in the search tree increases the possibilities of

finding an ultimately optimized tree structure, but simultaneously it tends to add exponentially the computational requirements for constructing the search tree. The designer of the algorithm must make a compromise in selecting the number of allowable stride lengths and the maximum number of levels so that on one hand the search tree becomes optimal enough for performing adequately fast searches in an adequately small memory space, and on the other hand the task of constructing and maintaining the search tree does not reserve a prohibitively large portion of the available processing time.

Next we will discuss an advantageous hardware implementation of an embodiment of the invention. Fig. 8 is a schematic block diagram where a routing processor 801 has a bus 802 for communicating with a search tree module 803. The latter comprises a first SRAM memory 804, a second SRAM memory 805 and a logic block 806. Bidirectional three-state buffers couple these blocks fo the processor bus 802 so that the first SRAM memory 804 is coupled to the bus through a first buffer 814, the second SRAM memory 805 is coupled to the bus through a second buffer 815 and the logic block 806 is coupled to the bus through a third buffer 816. Typically the processor bus comprises a data bus and an address bus, although these are not separately shown in Fig. 8.

Fig. 9 is a more detailed view of the contents of the logic block 806. The data bus 802a is coupled to the input of an input register 901, and the address bus 802b is coupled to the input of an address decoder 902. The outputs of the input register 901 are coupled both to the first SRAM memory (not shown in Fig. 9) and to a shifter 903. From the first and second SRAM memories there are couplings to a controller 904, an output of which is coupled to the shifter 903. The output of the shifter 903 is coupled to the second SRAM memory (not shown in Fig. 9). A further output from the controller 904 is coupled to the input of an output register 905, the output of which is coupled to the data bus. A logic block of this kind is most advantageously implemented as an ASIC or application specific integrated circuit, but also certain programmable logical gate arrays may be used especially for testing and prototyping.

The direct couplings between the processor bus 802 and the SRAM memories 804 and 805 enable the processor 801 to transparently access the look-up tables stored in the memories. In general, the search tree module 803 has two functional modes, namely a search mode and an update mode. Searching means that the processor 801 gives a piece of address information taken from a data packet and expects to receive the appropriate output port number in return. Updating means that the processor

uses said transparent access to change the contents of the look-up table. Routing must be halted for the duration of updating. Basically it would be possible to provide an embodiment of the invention where the update mode would concern only a part of the look-up table (note the previously mentioned need to only re-optimize a maximum of one subtree of the highest level) so that routing to those addresses that appear in other parts of the look-up table could continue. However, such an embodiment would require a rather complicated command architecture between the processor and the search tree module.

In order to maximize functional efficiency it is advantageous if the choice between functional modes can be made without using a specific register to store information of the current mode. We propose that address decoding is used to implement such functionality. The SRAM memories 804 and 805 have different addresses than the input and output registers of the logic block 806 (the latter two have most advantageously the same address so that a separately given read/write signal indicates, which register the processor wishes to point at). When an address given by the processor points at the input or output registers 901 or 905, the address decoder 902 interpretes it as a command for entering the search mode. If the processor points at the SRAM memories 804 or 805, the address decoder 902 generates the proper chip select signals to set the search tree module into the update mode.

The operation of the logic block 806 is described with reference to Fig. 10. When a packet or datagram arrives at an input buffer (not shown) of the processor 801, the header of the datagram is extracted and the processor sends the destination address or DA to the input register 901 by writing the DA onto the data bus 802a and the address of the input register 901 (together with a write signal) onto the address bus 802b. The address decoder 902 notes the address of the input register 901 on the address bus and gives a chip select signal to the input register 901. This procedure constitutes the beginning step 1001 in Fig. 10.

After a suitable time to allow the data bus to settle, e.g. after one clock cycle after receiving the address strobe signal, the input register 901 reads the contents of the data bus, i.e. the DA, at step 1002. Immediately thereafter is extracts as many most significant bits from the DA as the fixed stride length of the highest level determines and uses them to point at step 1003 at the first SRAM memory 804 which contains the highest level (level 0) of the search tree. The controller 904 waits for the memory circuit to react (e.g. 20 ns, if 12 ns SRAM is used) before reading the contents of the addressed memory location at step 1004.

At step 1005 the controller 904 decides, whether the record read from the memory location contained a next-hop or a pointer. In making the decision the controller uses some predefined rule that concerns certain bit values of the record. Previously we discussed a case where a bit combination "00" at the most significant bits of the record indicated a next-hop and all other bit combinations indicated pointers associated with various stride lengths. The invention does not limit the selection of the rule which is used: any combinations of any of the bits in the record can be used.

If the record was found to contain a pointer, the whole record is sent to the shifter 903 which also receives the original DA (or at least those bits thereof which were not used to point at the first SRAM memory) from the input register 901. At step 1006 the shifter reacts to those bits (previously designated as id bits) in the record that determine the applicable stride length. According to the value of said bits the shifter selects one of steps 1007, 1008 or 1009 and uses both the pointer from the latest read record and the additional bits taken from the DA to compose a pointer to a memory location in the second SRAM memory. At step 1010 the shifter uses this new pointer to point at the second SRAM memory 805, where the rest of the search tree is stored.

Reading the contents of a memory location in the second SRAM memory 805 takes place again after a suitable delay to allow the memory circuit to react to the addressing. Conceptually the process returns now to step 1004, although the record now comes to the controller 904 from the second memory circuit and not from the first one. According to step 1005 a check is again made to find out whether the new record contained a pointer or a next-hop. Finding a pointer causes a new round through steps 1006, one of 1007, 1008 or 1009, 1010 and back to 1004. This loop is repeated until at step 1005 the record is found to comprise a next-hop, which is sent to the output register at step 1011. At this time the search tree module 803 sends a hit signal to the processor 801, which then addresses the output register 905 to read the retrieved next-hop.

It is not necessary to use exactly two memory circuits in the hardware implementation. The whole search tree may be stored into one memory circuit, in which case the output of the shifter 903 in Fig. 9 is coupled to the address input of the first (and only) SRAM memory like the output of the input register 901. It is also possible to have more than two memory circuits for example so that every level of the search tree is stored into a memory circuit of its own. However, the second highest and the lower levels in the search tree become increasingly sparse compared

to the topmost level, so in most cases a single second memory circuit is enough to house all other levels than the highest one. Having at least two memory circuits allows the search operations to be pipelined. In other words, a new search in the first memory circuit may be started while the previous search still continues in the second memory circuit. Additionally it must be noted that it is not imperative to use SRAM memories, although they represent the most advantageous alternative at the priority date of this patent application in terms of speed and reliability. The invention may be applied regardless of the memory technology used.

The invention has been described with references to packet addressing as defined in the Internet Protocol, especially IPv4. This should not be construed as a limitation to the applicability of the invention, because the invention is equally applicable to all such routing tasks where a piece of information concerning the further treatment of a packet must be rapidly retrieved when a piece of variable-length header information has been obtained from the packet.

In the examples above we assumed that the stride on a certain level is determined by mapping the id value from the record of the previous level into a set of allowable stride lengths. However, especially when a separate stride field is used in the record (see Fig. 4a), it is possible to simply use a value in the stride field the absolute magnitude of which is equal to the number of the bits in the stride. This approach has the drawback of possibly wasting bit positions in the record, because it is usually most advantageous to have only a limited number of allowable stride lengths which are more effectively indicated through a mapping scheme.

The separately described features of the invention are basically usable in all combinations unless explicitly otherwise stated.

Claims

1. A method for searching through a variable stride level compressed search tree the nodes of which are represented by records in a memory, **characterized** in that the stride length (402, 404) to be used in advancing to a certain lower level from a certain node in the search tree is read from the record (502, 601, 603) that represented said node.

2. A method according to claim 1, **characterized** in that, for retrieving a piece of information concerning the further treatment of a data packet on the basis of a piece of address information contained within the packet, it comprises the steps of
a) addressing (1003) a first memory location (502, 601) with a first part of the address information (501),
b) reading (1004) a first record from the first memory location (502, 601),
c) deciding (1005), whether the first record contains a piece of information concerning the further treatment of the data packet or a pointer to a memory location,
d1) as a response to a decision that the first record contains a piece of information concerning the further treatment of the data packet, returning (1011) this piece of information and
d2) as a response to a decision that the first record contains a pointer to a memory location, taking a second part of the address information, combining (1007, 1008, 1009) the pointer with the second part of the address information into a combination and addressing a second memory location with the combination;
wherein step d2) comprises the substep of deciding (1006) the size of the second part of the address information on the basis of a first indicator value read from the first record.

3. A method according to claim 2, **characterized** in that step c) comprises the substep of reading said first indicator value (404) from the first record and deciding, on the basis of said first indicator value, whether the first record contains a piece of information concerning the further treatment of the data packet or a pointer to a memory location.

4. A method according to claim 2, **characterized** in that
- step c) comprises the substep of reading a second indicator value (401) from the first record and deciding, on the basis of said second indicator value, whether the first record contains a piece of information concerning the further treatment of the data packet or a pointer to a memory location, and

- step d2) comprises the substep of reading said first indicator value (402) from the first record.

5. A method according to claim 2, **characterized** in that step d2) comprises the substep of deciding the size of the second part of the address information by mapping said first indicator value into a set of allowed sizes for the second part of the address information.

6. A method according to claim 2, **characterized** in that step d2) comprises the substep of deciding the size of the second part of the address information by taking as many bits from the second part of the address information as the absolute magnitude of said first indicator value.

7. A method for constructing a variable stride level compressed search tree the nodes of which are represented by records in a memory, **characterized** in that it comprises the step of storing the stride length (402, 404) to be used in advancing to a certain lower level from a certain node in the search tree into the record that represents said node.

8. A method according to claim 7, **characterized** in that, for constructing a search tree for retrieving pieces of routing information concerning the further treatment of data packets on the basis of pieces of address information contained within the packets, it comprises the steps of

a) grouping (705) the pieces of routing information into subtree lists so that within each subtree list the pieces of routing information share a number of equally valued most significant bits,

b) representing (713) the equally valued most significant bits of the pieces of routing information within a subtree list with an inner node of the search tree to be constructed,

c) selecting (708) a stride length to be applied in advancing to the next lower level from the inner node and

d) placing the pieces of routing information within the subtree list into a subtree under the inner node, the number of branches in the subtree being determined by the selected stride length;

and that it additionally comprises the step of storing the selected stride length in the form of an indicator value (402, 404) into the record that represents the inner node.

9. A method according to claim 8, **characterized** in that it comprises the step of storing, into the record that represents the inner node, an indicator value (404)

which identifies the inner node as an inner node and simultaneously indicates the selected stride length.

10. A method according to claim 8, **characterized** in that it comprises the step of storing, into the record that represents the inner node, a first indicator value (401) which identifies the inner node as an inner node and a second indicator value (402) that indicates the selected stride length.

11. A method according to claim 8, **characterized** in that steps c) and d) comprise the substeps of

c1) selecting (708) a candidate stride length to be applied in advancing to the next lower level from the inner node and

d1) placing (713) the pieces of routing information within the subtree list into a subtree under the inner node, the number of branches in the subtree being determined by the selected candidate stride length;

and that the method additionally comprises the steps of

- repeating steps c1) and d1) for a number of different candidate stride lengths (715),
- associating the candidate stride length used and a parameter value describing the consumption of resources with each subtree constructed in step d1),
- selecting (716) that subtree to be part of the constructed search tree for which the associated parameter value describes the smallest consumption of resources and
- storing the candidate stride length associated with the selected subtree into the record that represents the inner node.

12. A method according to claim 11, **characterized** in that it comprises the step of recursively repeating (711, 712, 718, 720) the steps of claim 11 for all branches of the subtree under the inner node.

13. A device for searching through a variable stride level compressed search tree, comprising a memory (804, 805) and, stored within the memory, records (400) that represent the nodes of the search tree, **characterized** in that the memory comprises within at least a number of the stored records an integral part (402, 404) of the record that indicates a stride length to be used in advancing to a certain lower level from a certain node in the search tree represented by the record.

14. A device according to claim 13, **characterized** in that it comprises

- a microprocessor (801)
- a bus (802) coupled to said microprocessor,

- a search tree module (803) coupled to said bus,
- within the search tree module a memory circuit (804, 805) and
- within the search tree module a logic block (806);

wherein said memory circuit comprises, stored therein, the records (400) that represent the nodes of the search tree, and the logic block (806) is arranged to retrieve primary stored records from the memory circuit (804, 805) as a response to receiving pieces of address information from said microprocessor (801) and to use stride lengths based on indicator values read from the primary stored records in retrieving secondary stored records from the memory circuit (804, 805).

15. A device according to claim 14, **characterized** in that it comprises within the search tree module a first memory circuit (804) and a second memory circuit (805), and the logic block (806) is arranged to retrieve primary stored records from the first memory circuit (804) as a response to receiving pieces of address information from said microprocessor (801) and to use stride lengths based on indicator values read from the primary stored records in retrieving secondary stored records from the second memory circuit (805).

16. A device according to claim 15, **characterized** in that said bus comprises a data bus (802a) and an address bus (802b), and the search tree module comprises
- an input register (901) and an output register (905) coupled to said data bus,
- an address decoder (902) coupled to said address bus,
- a coupling from said input register (901) to said first memory circuit,
- a controller block (904),
- a coupling from said first memory circuit to said controller block (904),
- a shifter block (903),
- a coupling from said controller block (904) to said shifter block (903),
- a coupling from said shifter block (903) to said second memory circuit,
- a coupling from said second memory circuit to said controller block (904), and
- a coupling from said controller (904) to said output register (905);

wherein
- said address decoder (902) is arranged to decode an address on said address bus and to give a read signal to said input register (901),
- said input register (901) is arranged to read a piece of address information from said data bus as a response to said read signal and to extract a fixed number of bits from said address information and to address a first memory location within said first memory circuit with said extracted fixed number of bits,

- said controller block (904) is arranged to read a first record from said first memory location and to decide, on the basis of an indicator value within said first record, whether said first record contains a pointer or a piece of information concerning the further treatment of a data packet, and to couple records containing a pointer into said shifter block (903) and pieces of information concerning the further treatment of data packets to said output register (905),
- said shifter block (903) is arranged to extract a certain number of additional bits from the piece of address information read into said input register (901), where the number of additional bits depends on an indicator value within a record received from the controller block (904), and to address a second memory location within said second memory with a memory address composed of said additional bits and a pointer taken from a record received from the controller block,
- said controller block (904) is further arranged to read a second record from said second memory location and to decide, on the basis of an indicator value within said second record, whether said second record contains a pointer or a piece of information concerning the further treatment of a data packet, and
- said output register (905) is arranged to couple pieces of information concerning the further treatment of a data packet and received from said controller block (904) onto said data bus.

17. A router for constructing and using a search tree for retrieving pieces of routing information concerning the further treatment of data packets on the basis of pieces of address information contained within the packets, comprising
- a microprocessor (801)
- a bus (802) coupled to said microprocessor,
- a search tree module (803) coupled to said bus and
- within the search tree module a memory circuit (804, 805);
characterized in that the microprocessor is arranged to
a) group the pieces of routing information into subtree lists so that within each subtree list the pieces of routing information share a number of equally valued most significant bits,
b) represent the equally valued most significant bits of the pieces of routing information within a subtree list with an inner node of the search tree to be constructed,
c) select a stride length to be applied in advancing to the next lower level from the inner node and

d) place the pieces of routing information within the subtree list into a subtree under the inner node, the number of branches in the subtree being determined by the selected stride length;

and that the microprocessor is further arranged to store the selected stride length in the form of an indicator value into the record that represents the inner node within the memory circuit.

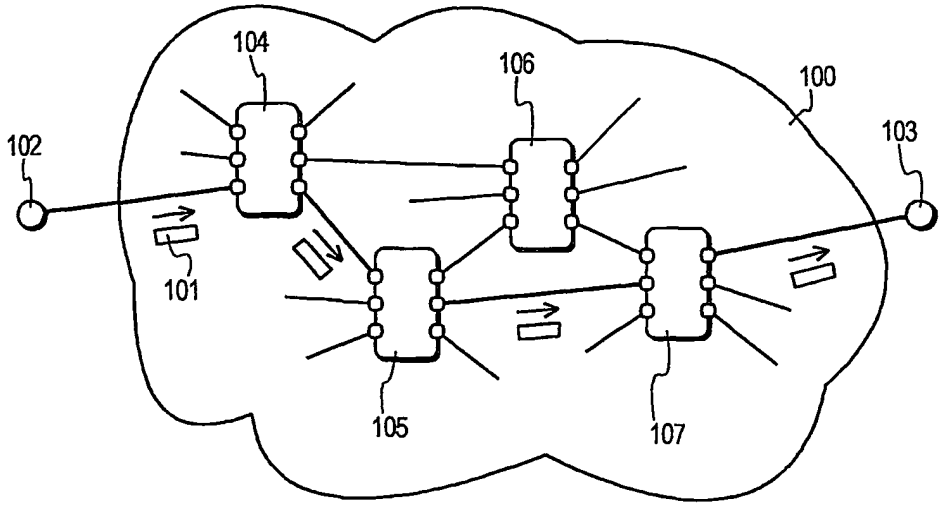


Fig. 1
PRIOR ART

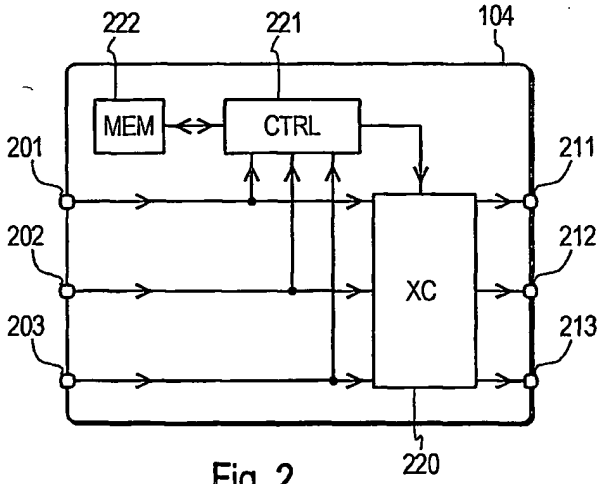


Fig. 2
PRIOR ART

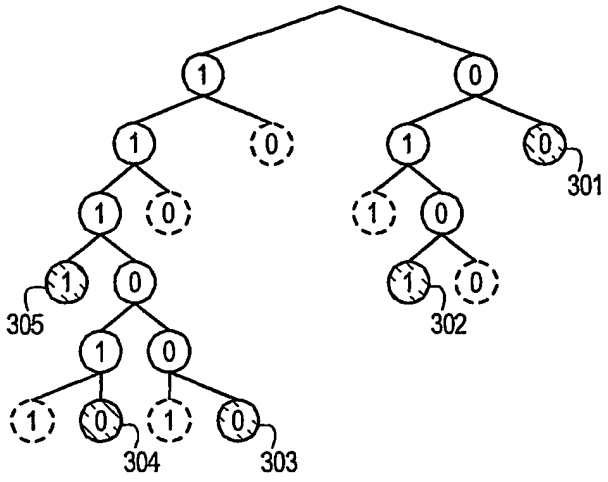


Fig. 3a
PRIOR ART

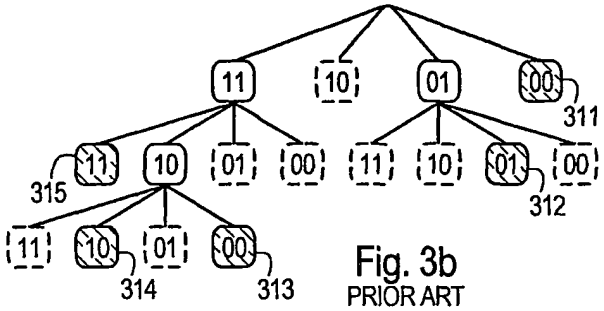


Fig. 3b
PRIOR ART

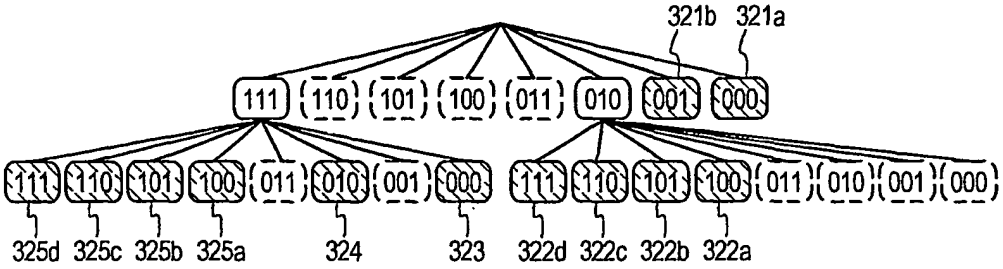


Fig. 3c
PRIOR ART

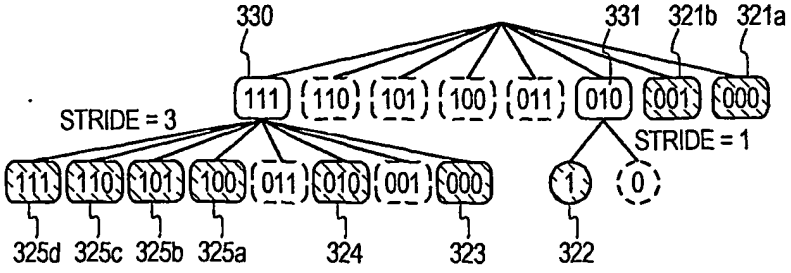


Fig. 3d
PRIOR ART

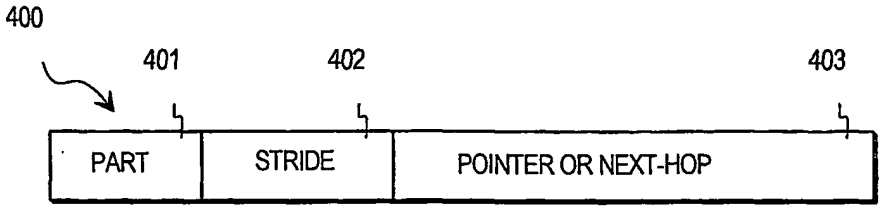


Fig. 4a

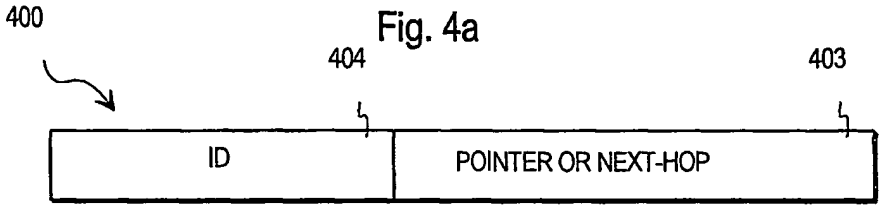


Fig. 4b

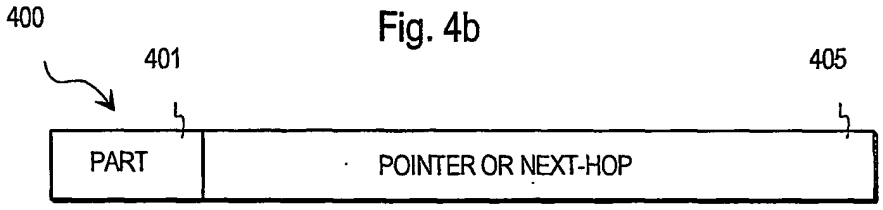


Fig. 4c

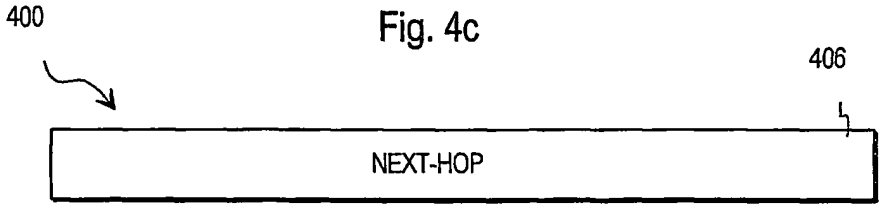


Fig. 4d

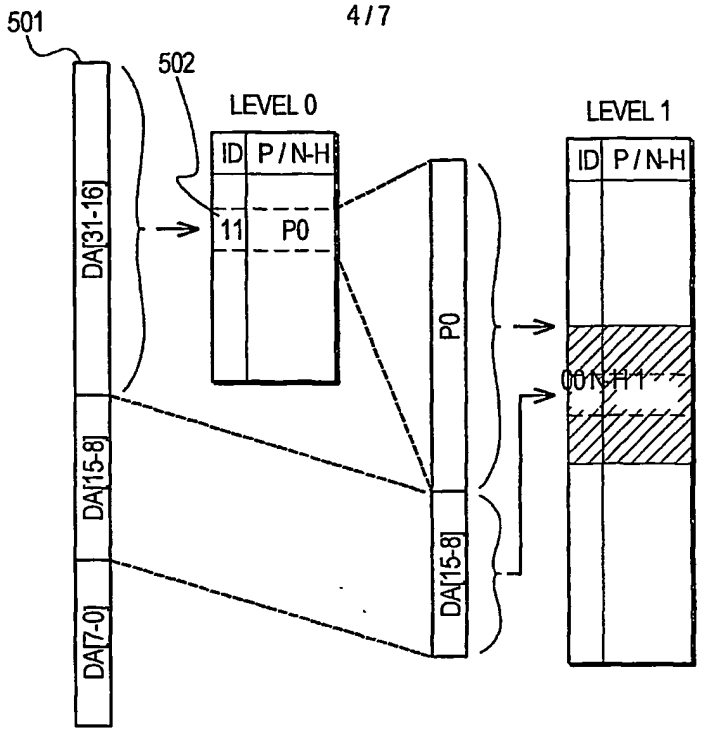

501
502
LEVEL 0
ID
P / N-H
11
P0
LEVEL 1
ID
P / N-H
DA[31-16]
DA[15-8]
DA[7-0]
P0
DA[15-8]
001 N-H 1

Fig. 5

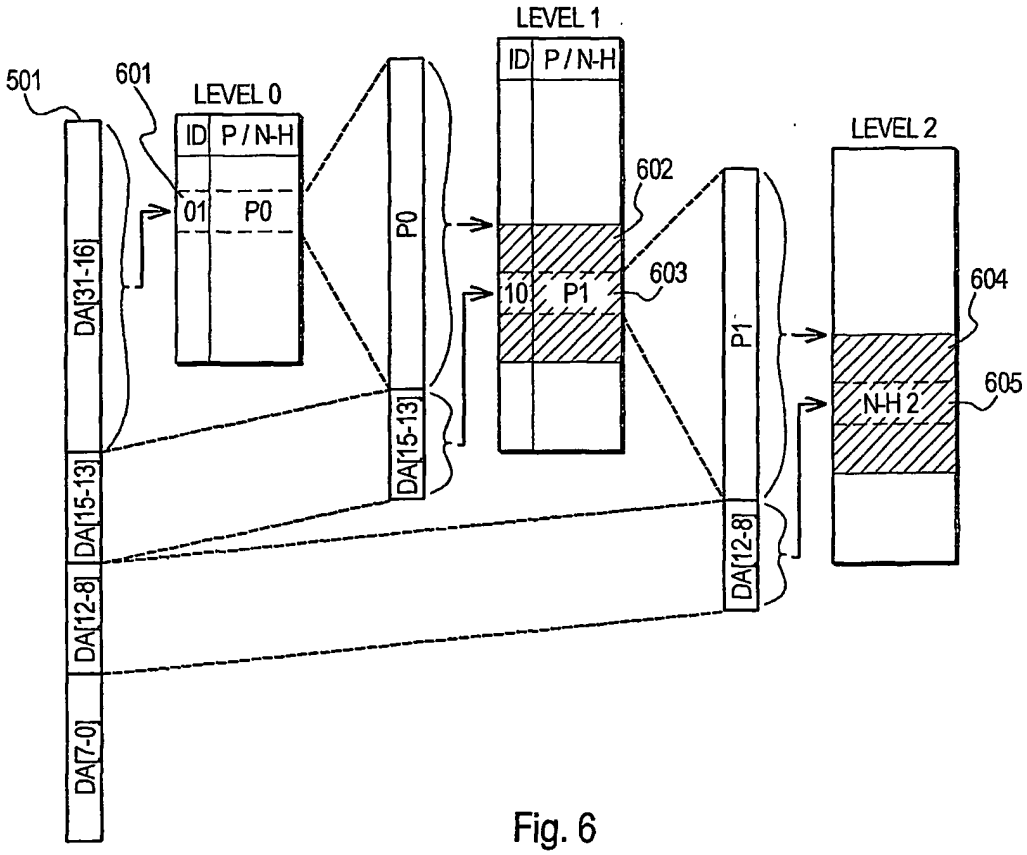

501
601
LEVEL 0
ID
P / N-H
01
P0
LEVEL 1
ID
P / N-H
602
10
P1
603
LEVEL 2
604
N-H 2
605
DA[31-16]
DA[15-13]
DA[12-8]
DA[7-0]
P0
DA[15-13]
P1
DA[12-8]

Fig. 6

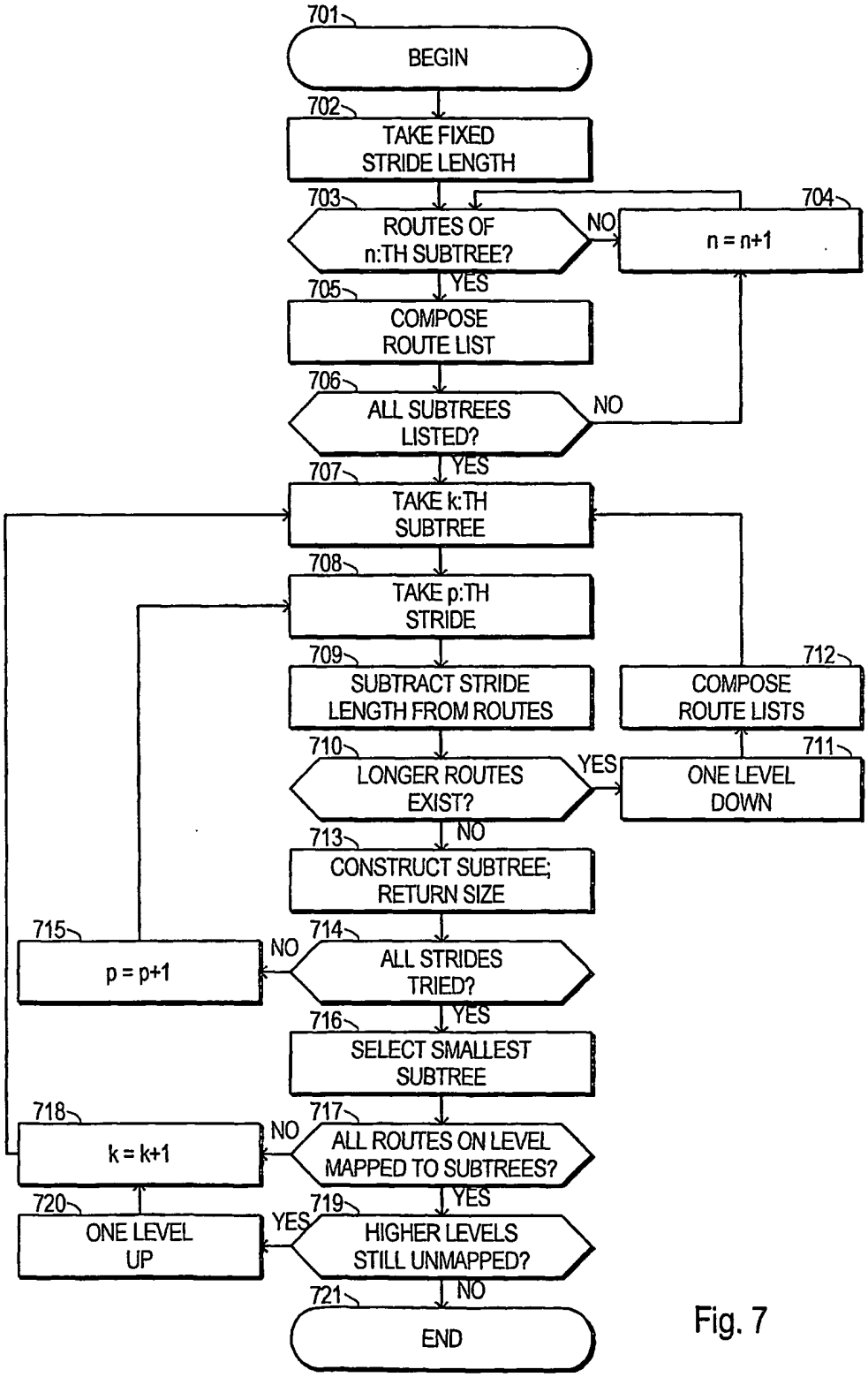

701
BEGIN
702
TAKE FIXED STRIDE LENGTH
703
ROUTES OF n:TH SUBTREE?
NO
704
n = n+1
YES
705
COMPOSE ROUTE LIST
706
ALL SUBTREES LISTED?
NO
YES
707
TAKE k:TH SUBTREE
708
TAKE p:TH STRIDE
709
SUBTRACT STRIDE LENGTH FROM ROUTES
712
COMPOSE ROUTE LISTS
710
LONGER ROUTES EXIST?
YES
711
ONE LEVEL DOWN
NO
713
CONSTRUCT SUBTREE; RETURN SIZE
715
p = p+1
NO
714
ALL STRIDES TRIED?
YES
716
SELECT SMALLEST SUBTREE
718
k = k+1
NO
717
ALL ROUTES ON LEVEL MAPPED TO SUBTREES?
YES
720
ONE LEVEL UP
YES
719
HIGHER LEVELS STILL UNMAPPED?
NO
721
END

Fig. 7

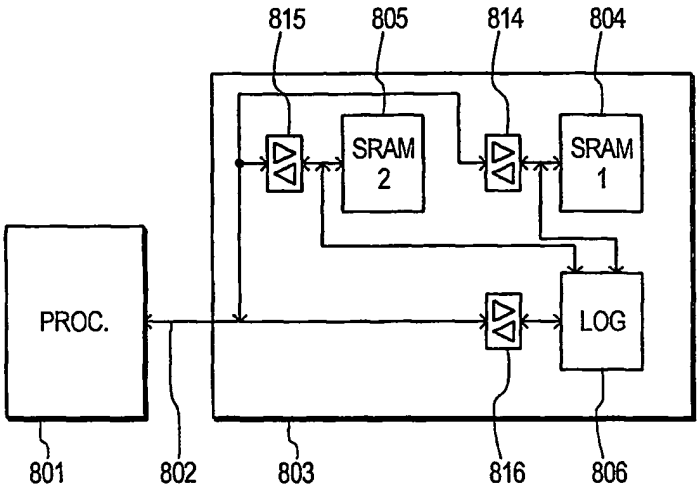

815
805
814
804
SRAM 2
SRAM 1
PROC.
LOG
801
802
803
816
806

Fig. 8

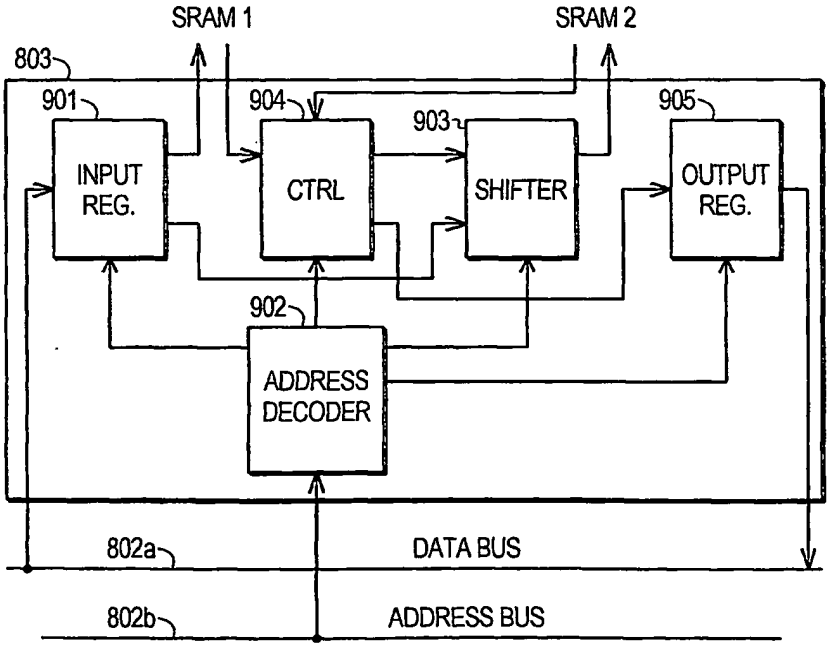

SRAM 1
SRAM 2
803
901
904
903
905
INPUT REG.
CTRL
SHIFTER
OUTPUT REG.
902
ADDRESS DECODER
802a
DATA BUS
802b
ADDRESS BUS

Fig. 9

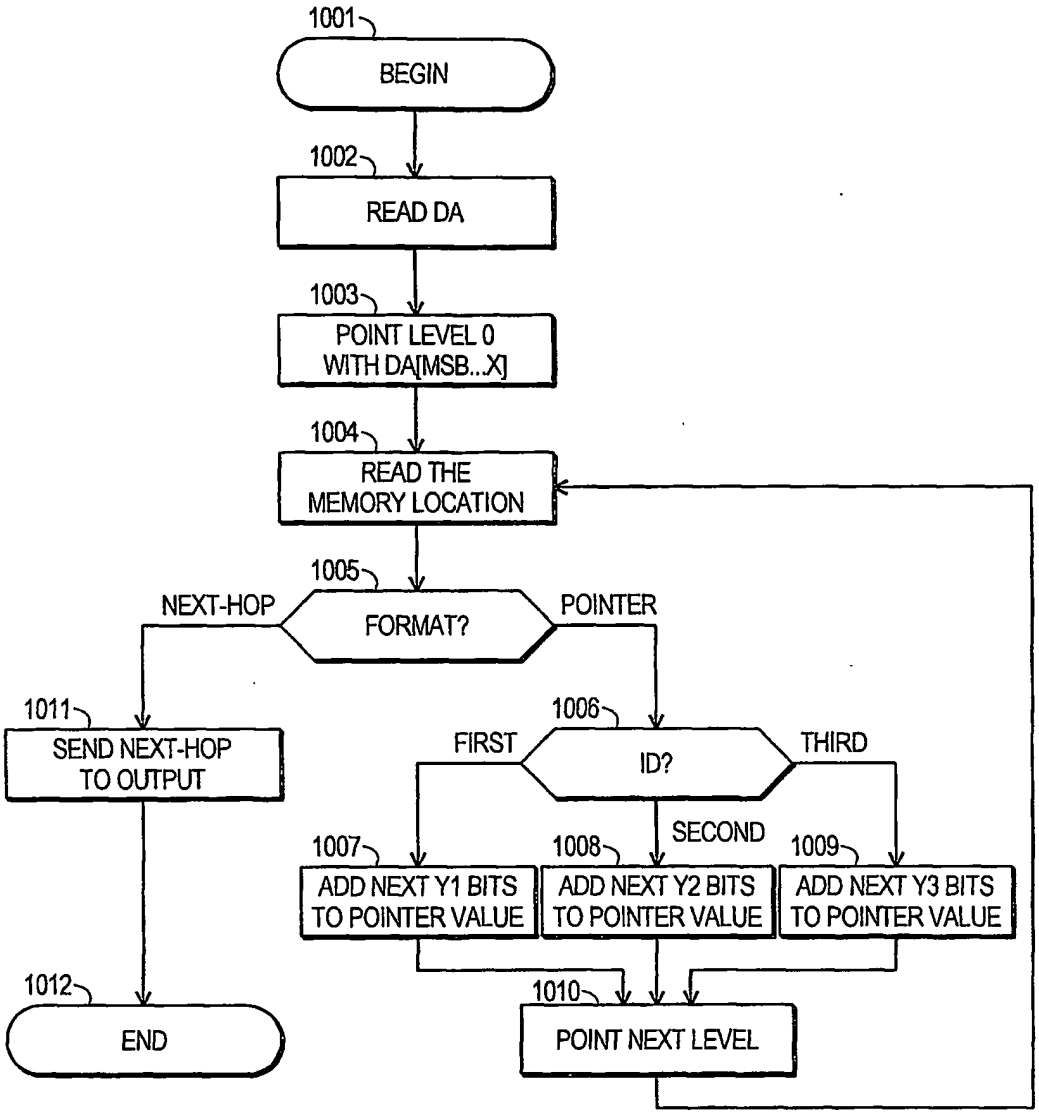

1001
BEGIN
1002
READ DA
1003
POINT LEVEL 0
WITH DA[MSB...X]
1004
READ THE
MEMORY LOCATION
1005
NEXT-HOP
FORMAT?
POINTER
1011
SEND NEXT-HOP
TO OUTPUT
1006
FIRST
ID?
THIRD
SECOND
1007
ADD NEXT Y1 BITS
TO POINTER VALUE
1008
ADD NEXT Y2 BITS
TO POINTER VALUE
1009
ADD NEXT Y3 BITS
TO POINTER VALUE
1012
END
1010
POINT NEXT LEVEL

Fig. 10

INTERNATIONAL SEARCH REPORT

International Application No
PCT/FI 01/00163

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 H04L12/56

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
INSPEC

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	NILSSON S ET AL : "IP-Address Lookup Using LC-Tries" IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, vol. 17 , no. 6 , June 1999 (1999-06), pages 1083-1092, XP002901790 page 1084, column 1, line 33 -page 1087, line 37 abstract	1,7,13, 17
Y	--- -/--	2-6, 8-12, 14-16

[X] Further documents are listed in the continuation of box C.

[ ] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
29 June 2001	19. 07. 2001

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Marianne Norrgren

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

International Application No
PCT/FI 01/00163

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	GUPTA P ET AL: "Routing Lookups in Hardware at Memory Access Speeds" INFOCOM '98 SEVENTEENTH ANNUAL JOINT CONFERENCE IEEE COMPUTER AND COMMUNICATIONS SOCIETIES. PROCEEDINGS IEEE 1998, vol. 3, 29 March 1998 (1998-03-29) - 2 April 1998 (1998-04-02), pages 1240 -1247, XP002901791 page 1240, column 1, line 22 -page 1241, column 1, line 13 page 1241, column 2, line 11 -page 1242, column 2, line 13 page 1243, column 1, line 18 -page 1245, column 1, line 23 figures 2-7 ---	2-6, 8-12, 14-16
X	RUIZ-SÂNCHEZ M A ET AL: "Survey and Taxonomy of IP Address Lookup Algorithms" IEEE NETWORK, vol. 15, no. 2, March 2001 (2001-03) - April 2001 (2001-04), pages 8-23, XP002901792 page 14, column 1, line 14 -page 17, column 1, line 52; figures 12-15 ---	1-17
A	CHEUNG G ET AL: "Optimal Routing Table Design for IP Address Lookups under Memory Constraints" INFOCOM '99 EIGHTEENTH ANNUAL JOINT CONFERENCE IEEE COMPUTER AND COMMUNICATIONS SOCIETIES. PROCEEDINGS IEEE 1999., vol. 3, 21 - 25 March 1999, pages 1437-1444, XP002901793 the whole document -----	1-17